AWS SERVICES:-

**Compute** – EC2, Lambda

**Storage** – S3, EBS ,EFS

**Database** – RDS, ✅ MySQL ✅ PostgreSQL ✅ MariaDB ✅ Microsoft SQL Server ✅ Oracle

**Networking** – VPC

**Security** – IAM

**Monitoring** – CloudWatch

**Teraform**

**To revoke multiple user**

**IAM:-not region specific its global**

-IAM (Identity and Access Management) is a AWS service it helps you securely control access to your AWS resources.

-With IAM, you can manage users on your AWS account, IAM allows you to create users, manage them, and define their permissions and access levels

**Key functionalities of IAM include:**

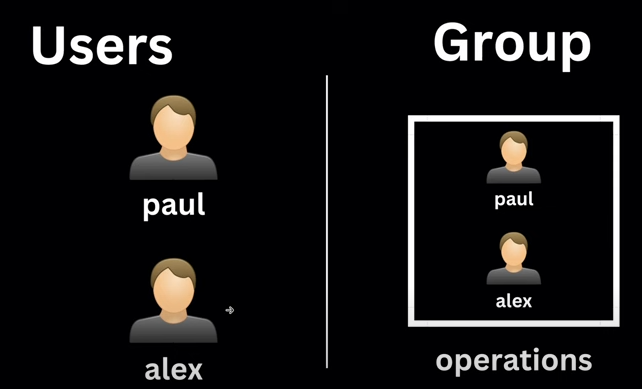
**Creating users** -You can create individual user accounts for people who need access to your AWS resources.

**Creating groups** – You can group users together and assign permissions to the group, making management easier for multiple users.

**Define Policies** – You can create and attach custom policies to define fine grained permissions for controlling access to AWS resources.

**Create Roles** – You can create roles to assign temporary permissions to AWS services or users, especially useful for securely managing permissions across different Aws resources.

**Manage Federated Access** – IAM allows integrating with external identity providers (like active directory) for centralized management of user access across AWS.



**Assigning permissions** – You can assign specific permissions to user, groups, or roles to control what actions they can perform on AWS services.

What is an IAM policy

An IAM policy is a JSON documents that defines permissions.

It consists of statements containing an effect (Allow/Deny) ,action ,resources and optional conditions.

**Create a User in AWS**

-Open IAM

-Go to “Users”

-Click “Add users”

-Enter User Details

* User name: Choose a unique name (e.g., dev-user).
* Provide access to AWS console
* Select I want to create an IAM user
* Console Password <select custom password>
* Access type:
  + Programmatic access: For CLI/SDK access (creates access key ID and secret).
  + AWS Management Console access: For console login (creates username & password).

-Set Permissions

You can give permission directly or else you by group but group should be exist if not then 1st create one group

Group name – Admin

Select Permission <Administrator access> that is in json format



You can create your won permission as well by clicking **create policy** over there you can go for **visual / Json**

**In Visual you can select what all service you need**

**Action allowed – Allow**

**Resources – All**

**Next**

**Policy name --**

Add user to group: Select a group with predefined policies.

Copy permissions from existing user.

Attach policies directly: Choose permissions like AmazonS3ReadOnlyAccess, etc.

**-**Review all info and click “Create user”.

-Download Credentials with .csv file.

Go to security credential over there you can see console sign in link **Copy That link open a new console and try to paste that link over there we can check like if we can access or not?**

Difference between IAM User & IAM Role

**IAM User**

* Represents a person or service in your AWS account.
* Has permanent credentials (username/password and access keys).
* Used for direct interaction with AWS services — for example, a developer logging in to the AWS Console or using the AWS CLI.
* You can assign policies directly to a user to control what actions they can perform.

**Example Use Case:**

A DevOps engineer who logs into the AWS Console to manage EC2 instances.

**🎭 IAM Role**

* Does not have permanent credentials.
* Assumed temporarily by users, services, or applications.
* Used to delegate access, especially across different AWS services or even AWS accounts.
* Roles have trust policies defining who can assume the role.

**Example Use Case:**

An EC2 instance assumes a role to get permission to read data from an S3 bucket without needing to store credentials.

**Networking:-**

VPC ( Virtual Private Cloud )

It is a Private isolate network within the AWS cloud where you can launch and manage your resources securely.

Why we need VPC

To Securely isolate and control network environments.



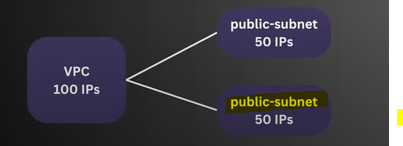
When we are creating a VPC we have to provide a CIDR block of IP address

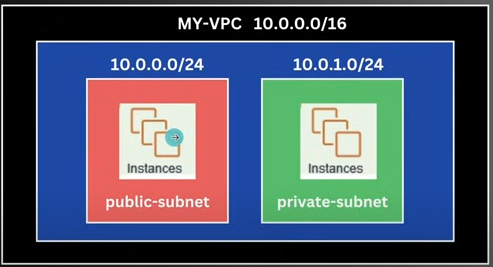
CIDR – (Classless Inter domain routing ) is a method for allocating IP address and routing internet protocol (IP ) packets.

**CIDR block size must be between /16 and /28.**

Subnets

A subnet is a smaller, segmented part of a large network that isolates and organizes devices within a specific IP address range.

Private Subnet

We can use VPC directly we have to create the resources inside the subnet.

**What happen when creating subnets**

**CIDR Block Allocation:-**

You specify a range of IP address (CIDR block) within the VPC’s IP address range for the subnet.

This determines the pool of IP addresses available for instances in the subnet.

Route Table:-

A route table is a set of rules, called routes, that are used to determine where network traffic from your subnets or gateway is directed. Each subnet in your VPC must be associated with a route table, which controls the routing for the subnet.

Internet Gateway:-

An internet gateway is a component that allows communication between instances in your VPC and internet.

NAT Gateway

* Enables private subnet instances to access the internet securely (e.g., for updates).
* NAT Gateway (managed AWS service) is preferred over NAT Instance for scalability.

Security Groups:-

Network firewall rules that control inbound and outbound traffic for instances.

It operates at the instance level (not at the subnet level like NACLs).

**Inbound Rules (Incoming Traffic)**

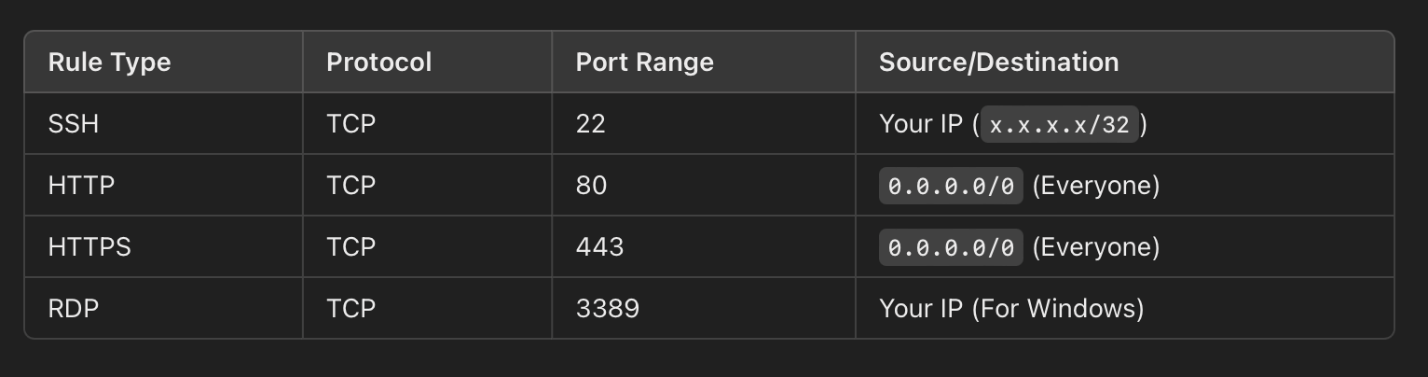
Controls what traffic can enter the instance.

Example: Allow SSH (22) for remote access.

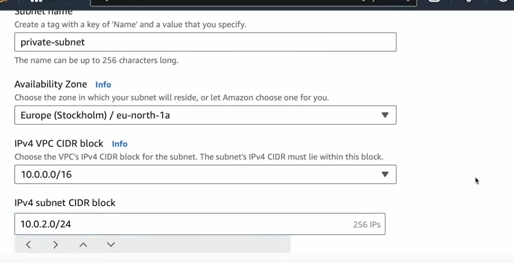
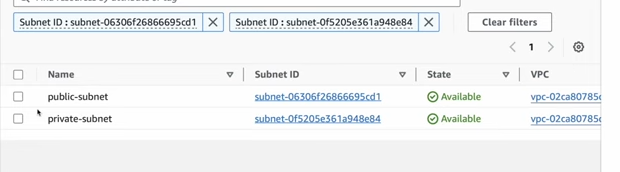
**Outbound Rules (Outgoing Traffic)**

Controls what traffic can leave the instance.

By default, all outbound traffic is allowed.



To Create VPC:-

* Go to console search VPC
* Create VPC
* Choose **“VPC only”**
* **Name tag**: my-vpc
* **IPv4 CIDR block**: (e.g., 10.0.0.0/16) here you have to give the range between 16 -28.
* Enable IPv6 CIDR block (for IPv6 support)
* Choose **Tenancy** (**default** – shared hardware & **Dedicated** means own hardware)
* Click **“Create VPC”**.
* Go to subnet
* Create Subnets
* Select your newly created VPC. My-vpc
* Define one or more subnets (e.g., 10.0.1.0/24, 10.0.2.0/24) and assign them to different **Availability Zones**.
* Create subnet
* 
* 

Create Internet Gateway

-Go to **Internet Gateways** > Click **“Create internet gateway”**.

-Name it (e.g., MyIGW) and create.

-**Attach** it to your VPC.

**-Update Route Tables**

1. Go to **Route Tables**, find the one associated with your VPC.
2. Edit routes > Add route:
   * **Destination**: 0.0.0.0/0
   * **Target**: Your **Internet Gateway**
3. Associate this route table with your **public subnet(s)**.

**-Configure Security Group & Network ACLs (optional)**

Security Groups control inbound/outbound traffic to EC2 instances. Set them as needed for your application (e.g., allow SSH, HTTP).

What is EC2 Instance:-

* EC2 stands for (Elastic Compute Cloud)it is a service that provides scalable virtual servers in the cloud. It allows users to run applications on virtual machines (instances) with different configurations of CPU, memory, and storage. It enables you to **scale these instances as needed.**
* over there we can check the instances
* Like we have one application terraform system admin by connecting that we can check the session manager, ssh client, EC2 serial console.
* After that we can connect so that the instance will open now you can execute any type of command over there.

**What is an instance**

An instance is a **virtual server launched using the EC2 service**

It includes **compute, memory, storage, and networking capabilities**

How to secure EC2 Instance

To secure EC2 instance by using security groups, which act as virtual firewalls, and network ACL (Access Control Lists) to control incoming and outgoing traffic.

You can use IAM roles to control permissions and enable encryption for data at rest and in transit.

EC2 Instance Metadata

It's information about the running EC2 instance

It includes details like VPC, subnet, public IP, private IP, and security groups.

**Creating an linux instance on AWS**

**Step 1: Log in to the AWS Management Console**

Go to the [AWS Management Console](https://aws.amazon.com/console/).

Log in with your AWS account.

**Step 2: Navigate to EC2**

From the Console home, select EC2 from the Services menu or search for EC2.

**Step 3: Launch an Instance**

On the EC2 Dashboard, click on Launch Instance.

Click Launch Instance again to start the setup process.

**Step 4: Choose an Amazon Machine Image (AMI)**

Select an AMI that suits your needs (e.g., Amazon Linux 2, Ubuntu, Windows).

**Step 5: Choose an Instance Type**

Choose an instance type based on your requirements, such as t2.micro (Free Tier eligible) or another type depending on your workload.

Click Next to configure instance details.

**Step 6: Configure Instance Details**

Set the Number of Instances you want to launch.

Select a Network (VPC) and Subnet if you have specific requirements. The default VPC and subnet are usually sufficient.

Configure Auto-assign Public IP .

Other settings like IAM role, Shutdown behavior, and Monitoring can also be configured as needed.

Click Next: Add Storage.

**Step 7: Add Storage**

By default, an Amazon EBS volume is added. You can adjust its size and type as per your requirements.

Add additional volumes if required.

Click Next: Add Tags.

**Step 8: Add Tags (Optional)**

Click Add Tag, enter a Key (e.g., Name) and Value (e.g., MyEC2Instance) to label your instance.

Tags help with organizing and managing your AWS resources.

**Step 9: Configure Security Group**

Either select an existing security group or create a new one.

Define rules to allow traffic (e.g., allow SSH on port 22 for Linux instances or RDP on port 3389 for Windows).

Set the source to a specific IP (your IP or a custom IP range) to restrict access.

Click Review and Launch.

**Step 10: Review and Launch**

Review all your instance settings.

Click Launch.

**Step 11: Choose or Create a Key Pair**

Select an existing key pair or create a new one. Download the key pair (.pem file) and store it securely. This will allow you to SSH into the instance.

Confirm you’ve downloaded the key pair, then click Launch Instances.

**Step 12: Access Your Instance**

Once the instance is in a running state, click View Instances.

Select your instance and note the Public IPv4 address or DNS name.

For Linux instances, use SSH to connect:

bash

ssh -i /path/to/your-key.pem ec2-user@your-public-ip

Create windows instances

-EC2 dashboard

-Launch Instance

-Configure the Instance - Give your instance a name (like My-Windows-Server)

-**Application and OS Image (AMI):**

* Click **"Browse more AMIs"**.
* Search for **"Windows Server"** (e.g., Windows Server 2022, 2019, etc.).
* Choose a version you want (AWS has official AMIs for Windows Server).

-Choose Instance Type

Choose an instance type depending on how powerful you need it.

* Example: t2.micro (Free tier) or t3.medium (for more power).

-Key Pair (Login) / **Create a new key pair** (or use an existing one).

-Network Settings

. Choose or create a VPC and Subnet (default is fine for beginners).

. Allow RDP (Remote Desktop Protocol) traffic

.source type – MY IP

. Add a rule to allow **port 3389** (for Windows RDP).

-Configure Storage

30 GB is usually good for a basic server, modify if you need more space. By clicking add volumn

-Click **Launch Instance**!

-Connect to the Instance

. Select your instance → click **Connect** → choose **RDP client**.

.Download RDP File

When we double click that file it will launch from the local application /Remote desktop application

. **public IPv4 address**.

. Your **key pair** to get the Windows User name – **administrator**

**To get the password click on Get password** upload your temp file click on Decrypt password

. Download the **RDP file** and open it with Remote Desktop Connection.

AMI (Amazon Machine Image):-

-An AMI is a pre-configured virtual machine image used to create EC2 instances (e.g., Amazon, Linux, Ubuntu, Windows).

-It includes the operating system, application software and any configurations.

**Types of AMI**

-Public AMI- Available to all AWS users. Useful for basic use cases like popular operating systems e.g.- Ubuntu

Private AMI – Created by a user and only available within that account or shared with specific account.

Paid AMI – Provided by third parties through AWS marketplace, offering software like database, web servers, or pre- configured environments.

**How to create image in aws**

-Go to the **EC2 Dashboard**.

-**Image and templates** → **Create Image**.

-**Image Name:** Give your AMI a name (example: MyWindowsServerBackup).

-**Image Description:** (Optional) Add details for your reference.

-If you want the instance to stay online without restarting, you can check **"No reboot"**.

-Create Image

**Use Your New Image**

* Once the image is ready (**status = available**):
  + You can **launch new instances** directly from this AMI.
  + You can **copy** it to other regions.
  + You can even **share** it with other AWS accounts if needed.

Elastic IP address

* An Elastic IP address is a static IP address designed for dynamic cloud computing.
* With an EIP, you get a permanent IP address that you can:
* Attach to any EC2 instance
* Detach and reassign to another EC2 instance if needed
* This makes sure your server is always reachable with the same IP, even if you reboot it.

Handson:-

-Go to AWS Console → EC2 → Elastic Ips

-Click Allocate Elastic IP address.

-Scope: Amazon pool (default option). Click Allocate.

-Select the EIP you just created -> lick Actions → Associate Elastic IP address.

-Choose: Resource type: Instance -> Select your EC2 instance (Ubuntu or Windows)

-(Optional) Choose a specific network interface if needed. ->click Associate

-Go to EC2 → Instances -> Select your instance -> Under Public IPv4 address, you should now see your Elastic IP assigned.

Auto Scaling

You can use Auto Scaling to manage the scaling up or scaling down the EC2 instances.

Auto-scaling is a function that allows you to provision and launch new instances whenever there is a demand. It allows you to automatically increase or decrease resource capacity in relation to the demand.

**-Auto Scaling Group (ASG)** – A collection of EC2 instances managed together.

**Launch Template-**Define instance type, AMI, Security group and other configuration.

**Scaling Policies-**Determines when to add or remove instances. (e.g.- CPU usage exceeds 70%)

Handson:-

* Create a Launch Template
* Go to the AWS Management Console → Search EC2.
* In the EC2 Dashboard, click Launch Templates (left panel).
* Click Create launch template.
* Enter a Template name (e.g., MyAutoScalingTemplate).
* Select an Amazon Machine Image (AMI) (e.g., Amazon Linux 2).
* Choose an Instance type (e.g., t2.micro).
* Configure a Key pair for SSH access.
* Under Network settings, select an existing security group or create a new one

Here we are creating app load balancer so click on allow http traffic

* No of instance -2
* Advance details << In user data add the script

#!/bin/bash

sudo yum update -y

# Install Apache web server (httpd)

sudo yum install -y httpd

sudo systemctl start httpd

sudo systemctl enable httpd

# Create a simple HTML file to verify the web server is running, including

dynamic hostname

echo "<html><h1>Welcome to Apache Web Server on Amazon Linux -

$(hostname)!</h1></html>" > /var/www/html/index.html

* Click Create launch template.

Load Balancer:-

AWS Load Balancer is a managed service that distributes incoming traffic across multiple targets (like EC2 instances, containers, or IP addresses) to ensure high availability, fault tolerance, and better performance.

**Types of AWS Load Balancers**

**-Application Load Balancer (ALB)** :- Most commonly used for HTTP and HTTPS based websites and applications.... It works at OSI Layer 7

* **Network Load Balancer (NLB) -** Designed for TCP, UDP, and TLS protocols. It offers high performance and low latency, suitable for applications like gaming or financial trading where time is critical. It works at OSI Layer 4
* **Gateway Load Balancer (GWLB)** - Used for deploying and managing third-partyvirtual appliances like firewalls and monitoring solutions
* **Classic Load Balancer (CLB) –** It *is an older, previous-generation type*

Handson-

Creating a **Application Load Balancer**

* Once the template is created select that template (e.g., MyAutoScalingTemplate)
* Version <Default(1)>
* Next
* Availability zone selects eu-north-1a,1b,1c
* Next
* Attached to an existing load balancer
* Select the target group
* Health check click on ELB
* Health check period e.g-300sec
* Next
* Desired capacity 2 minimum
* Scaling limit <min 1 , max 3>
* Target tracking policies <Target group><Cpu utilization , 80 , warmup 300 sec>
* Next
* Review
* Create auto scaling group

Steps to Create ASG :-

* Launch Template or Configuration
* Create Auto Scaling Group
* Select VPC and Subnets
* Attach Load Balancer (Optional)
* Configure Scaling Policies
* Health Checks
* Add Notifications (Optional)
* Review and Create

AWS RDS (Relational Database Service)

RDS as a managed database service that simplifies database setup, operation, and scaling.

Instead of manually installing and maintaining a database, RDS automates tasks like backups, patching, and scaling.

✅ MySQL ✅ PostgreSQL ✅ MariaDB ✅ Microsoft SQL Server ✅ Oracle ✅ Amazon Aurora (AWS's own database)

* Create an RDS MySQL instance.
* Connect to it from an EC2 instance.
* Perform basic database operations.

**RDS Instance**

-Search RDS << Database << Create Database << standard create << Mysql

-version : Choose the latest version.

-Use Free Tier <Production /Dev/Test>

-Instance name <database-1>

-Self managed << Username will be 'admin' and you can set password (you can't use

special character)

-Connectivity << Don’t connect to EC2

-VPC select

-Keep the Public access to True to access it from Local or remote server

-Create new VPC << Mysql sg << Create a security group (and allow 3306 from everywhere)

-Storage: Storage Type: General Purpose (SSD) Allocated Storage: 20 GB Enable Auto Scaling: No

-Availability zone select

-Password authentication

-After creating, you can find Endpoint (hostname) to connect to this DB.

Then create EC2 Instance in the same VPC.

-Instance << Launch instance << name (demo Rds) << Amazon linux <<key pair without

-Network << Allow http

-click on **demo Rds** instance << Connect << EC2 << connect

-Install it

* Install MySQL client: sudo yum update -y && sudo yum install mysql -y
* Find RDS Endpoint: Go to AWS Console → RDS → Select your database. Copy the "Endpoint" (e.g., my-rds-instance.xyz.us-east-1.rds.amazonaws.com).
* Connect to RDS: mysql -h <RDS-ENDPOINT> -u admin -p Enter your password (MySecurePassword123!).

Show databases;

Use my\_app\_db;

Show tables;

Select \* from contacts;

What is CloudWatch?

The Amazon CloudWatch has the features like:

Depending on multiple metrics, it participates in triggering alarms.

Helps in monitoring the AWS environments like CPU utilization, EC2, Amazon RDS instances, Amazon SQS, S3, Load Balancer, SNS, etc.

EBS Snapshots:-

An EBS snapshots is a point-in-time copy of an EBS volume.

It can be used to create backups, migrate data between regions, or create new volumes.

Storage (EBS, EFS, S3)

S3 ((Simple Storage Service)

-AWS S3 is a cloud-based storage service that allows you to store, manage, and retrieve large amounts of data like files, images, videos, and backups securely and at scale.

-It provides highly reliable, scalable object storage, making your data accessible from

anywhere, anytime, via the internet.

Handson:-

✅ Step 1: Create an S3 Bucket

* Go to the S3 Console.
* Click Create bucket.
* Set: Bucket name: my-s3-demo-bucket-
* Region: Same as your EC2 instance
* Leave defaults or uncheck Block all public access if you need public files (not recommended for private data).
* Click Create bucket.

✅ Step 2: Launch Ubuntu EC2 Instance (if not already running)

* Go to EC2 Console.
* Launch Ubuntu Server 22.04.
* Choose t2.micro, same region as S3 bucket.
* Ensure port 22 (SSH) is open.
* Launch and note the Public IP and Key Pair.

✅ Step 3: Create an IAM Role for S3 Access

* Go to the IAM Console.
* Click Roles > Create Role.
* Select AWS service > EC2.
* Click Next and search for AmazonS3FullAccess or create a custom policy.
* Name the role EC2-S3-Access-Role
* Create the role.

✅ Step 4: Attach IAM Role to EC2 Instance

* Go to EC2 > Instances.
* Select your instance → Actions → Security → Modify IAM Role.
* Choose EC2-S3-Access-Role.
* Click Update IAM role.

✅ Step 5: Connect to EC2 and Install AWS CLI ssh -i your-key.pem ubuntu@<EC2-Public-IP> Then run:

**Install AWS CLI v2**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip" unzip awscliv2.zip sudo ./aws/install

**Test installation:**

aws --version You do not need to configure aws configure because the EC2 instance now has an IAM role with S3 permissions.

✅ Step 6: Upload and Download Files from S3

**Create a file**

echo "Hello from EC2 to S3" > testfile.txt

**Upload to S3**

aws s3 cp testfile.txt s3://my-s3-demo-bucket-<unique-id>/

**List contents of S3 bucket**

aws s3 ls s3://my-s3-demo-bucket-<unique-id>/

**Download from S3**

aws s3 cp s3://my-s3-demo-bucket-<unique-id>/testfile.txt downloaded.txt

**Check downloaded file**

cat downloaded.txt

🧼 Optional Cleanup

**Delete objects from S3:**

aws s3 rm s3://my-s3-demo-bucket-<unique-id>/testfile.txt

* Delete bucket from S3 Console.
* Terminate EC2 instance.
* Delete IAM Role if no longer needed.

EBS (Elastic block store) volumes introduction

EBS is a cloud-based storage service that provides durable, high-performance block storage for use with Amazon EC2 instance.

-It works like a virtual hard drive, allowing you to store and access data even when your EC2 instances are stopped or terminated.

IMP Points:-

-Region & AZ specific

-Build-in Redundancy

EBS volumes are automatically replicated within the same availability zone to prevent data lose due to hardware failures.

-Different volume types (gp2/3,io1/2,st1,sc1)

-Allow encryption & snapshot for backup

-Volume can be resizable , no data lose during resizing, no need to restart ec2 during the process.

✅ Step 1: Launch an EC2 Instance (if not already running)

* Go to the EC2 Dashboard.
* Click Launch Instance.
* Choose Amazon Linux 2 AMI.
* Select an instance type (e.g., t2.micro for free tier).
* In Configure Instance, select an Availability Zone (e.g., us-east-1a).
* Complete the launch and wait for the instance to be in running state.

✅ Step 2: Create an EBS Volume

* In the EC2 dashboard, go to Elastic Block Store > Volumes.
* Click Create Volume.
* Set:
  + Volume type: gp3
  + Size: e.g., 5 GiB
  + Availability Zone: Match your EC2 instance's AZ (e.g., us-east-1a)
* Optionally, add a Tag like:
  + Key: Name, Value: MyEBSVolume
* Click Create Volume.

✅ Step 3: Attach the EBS Volume to EC2

* Go to Volumes, right-click the newly created volume.
* Choose Attach Volume.
* Select the EC2 Instance ID.
* Device name: leave default (/dev/xvdf).
* Click Attach.

EBS Volume Resize: -

Go to the volume << Modify volume << size increase << Modify

To check

Go to terminal << lsblk << over there we can check its increased.

In Linux to increase the volume we are using LVM.

To copy EBS to different AZ.

Go to volume << select the volume << Action << Create snapshot << description << create snapshot.

Now go to snapshot now you can see the snapshot is added.

Create Volume from snapshot:-

Select the volume << create volume from snapshot << change the zone << create volume

EFS:-

**EFS (Elastic File System)** is a scalable and fully managed file storage service designed for use with AWS cloud services .It is built to provide shared, persistent storage for applications and workloads.

* For doing this activity we are creating a backup plan adding backup rule and creating a backup vault where the recovery id of the EFS is being generated.

Handson:-

* Go to EC2 Dashboard.
* Click Launch Instance.
* Choose: Amazon Linux 2 AMI.
* Instance type: t2.micro (for Free Tier).
* Select an Availability Zone (e.g., us-east-1a).
* Add a security group with ports 22 (SSH) and 2049 (NFS) open.
* Launch the instance.

✅ Step 2: Create a Security Group for EFS

* Go to VPC > Security Groups.
* Click Create security group:
  + Name: EFS-SG
  + Description: Allow NFS access
  + VPC: same as EC2 instance
* Add Inbound rule:
  + Type: NFS
  + Protocol: TCP
  + Port: 2049
  + Source: EC2 instance’s security group
* Click Create security group.

✅ Step 3: Create an EFS File System

* Go to Amazon EFS Console.
* Click Create file system.
* Choose:
  + VPC: same as your EC2 instance
  + Availability Zones: choose the one where EC2 is running
* Automatic mount targets will be created in each subnet (keep defaults).
* In File system policy and access, leave as-is or add tags if needed.
* Click Create.

✅ Step 4: SSH into Ubuntu EC2 and Install NFS Tools ssh -i your-key.pem ubuntu@<EC2-Public-IP>

**Update package list**

sudo apt update

**Install NFS client**

sudo apt install -y nfs-common

✅ Step 5: Mount the EFS File System Get the EFS File System ID (e.g., fs-abc12345) from the EFS console.

**Create a mount directory**

sudo mkdir -p /mnt/efs

**Mount the EFS using DNS name**

sudo mount -t nfs4 -o nfsvers=4.1 fs-abc12345.efs.<region>.amazonaws.com:/ /mnt/efs

**Example for us-east-1:**

sudo mount -t nfs4 -o nfsvers=4.1 fs-abc12345.efs.us-east-1.amazonaws.com:/ /mnt/efs

✅ Step 6: Verify the EFS Mount

**List files**

ls /mnt/efs

**Write a test file**

echo "EFS is mounted!" | sudo tee /mnt/efs/test.txt

**Check the content**

cat /mnt/efs/test.txt

CloudWatch?

Helps in monitoring the AWS environments like CPU utilization, EC2, Amazon RDS instances, Amazon SQS, S3, Load Balancer, SNS, etc.

Depending on multiple metrics, it participates in triggering alarms.

**CloudWatch Important**

* Real-time Monitoring: Helps in tracking resource utilization (CPU, Memory, Network, etc.).
* Log Aggregation & Analysis: Collects logs from AWS services and applications.
* Automatic Alarming: Can notify users when a metric crosses a threshold.
* Performance Optimization: Helps analyze trends and optimize resources.
* Troubleshooting & Debugging: Logs and alarms help in identifying system failures.

Handson:-

**Monitor CPU utilization of an EC2 instance**

**Step 1: Launch an EC2 Instance**

* Go to AWS Console → EC2 → Launch Instance.
* Select Amazon Linux 2 as the AMI.
* Choose an instance type (t2.micro for free-tier).
* Configure Security Group: Allow SSH (22) and HTTP (80).
* Launch the instance and connect via SSH.

**Step 2: Enable CloudWatch Monitoring**

* Navigate to EC2 Dashboard → Instances.
* Select the instance and go to the Monitoring tab.
* Click on View in CloudWatch.
* You’ll see default metrics like CPUUtilization, NetworkIn, NetworkOut, etc.

**Step 3: Create a CloudWatch Alarm**

* Go to CloudWatch Console → Alarms → Create Alarm.
* Select metric:
* Click Select metric → Choose EC2 → Per-Instance Metrics.
* Select CPUUtilization for your EC2 instance.
* Set conditions:
* Threshold: Greater than 70%.
* Period: 5 minutes.
* Configure actions:
* Choose SNS Notification.
* Create an SNS topic and subscribe with your email.
* Create Alarm.

**Test it: Install a CPU stress tool to generate load.**

sudo yum install stress -y stress --cpu 2 --timeout 300

Difference between **Security group and Network ACL**.

* -A security group acts as a firewall at the **instance level**, controlling inbound and outbound traffic.
* -A Network ACL operates at the **subnet level** and controls traffic entering or leaving a subnet.

Difference betwn On-demand, reserved and spot instances

**On-demand**- Pay-as-you go instances with no upfront commitment.

**Reserved** – Instances purchased with a 1- or 3-years commitment offering significant cost saving.

**Spot** – Instances with variable pricing based on supply and demand, suitable for workloads that can handle interruptions.

How do you upgrade or downgrade a system with near-zero downtime?

You can upgrade or downgrade a system with near-zero downtime using the following steps of migration:

Open EC2 console

Choose Operating System AMI

Launch an instance with the new instance type

Install all the updates

Install applications

Test the instance to see if it’s working

If working, deploy the new instance and replace the older instance

Once it’s deployed, you can upgrade or downgrade the system with near-zero downtime.

How many S3 buckets can be created?

Up to 100 buckets can be created by default.

What is AWS S3, and how does it fit into a DevOps environment?

Answer: AWS S3 is an object storage service that provides scalable, durable, and secure storage for a wide range of data. In a DevOps environment, S3 is used for storing application artifacts, backups, logs, and as a source for deploying code and assets.

Explain the concept of S3 buckets and objects. How is data organized in S3?

Answer: S3 uses a flat namespace where data is organized into buckets, which are top-level containers for storing objects. Objects are the actual data files stored within buckets and can range from small files to large data sets.

What are the storage classes available in S3, and when would you use each one?

S3 offers various storage classes, including STANDARD, INTELLIGENT\_TIERING, ONEZONE\_IA, GLACIER, and more. The choice depends on factors like data access frequency, durability requirements, and cost considerations. For example, STANDARD is used for frequently accessed data, while GLACIER is for long-term archival.

How can you secure data in S3 buckets, and what are the best practices for implementing data access controls?

Answer: Best practices for securing S3 data include:  
— Using bucket policies and ACLs to control access.  
— Enforcing encryption using server-side encryption.  
— Utilizing IAM roles and policies for fine-grained access control.  
— Enabling versioning and MFA Delete for data protection.

Explain the significance of S3 Access Control Lists (ACLs) and S3 Bucket Policies for fine-grained access control.

Answer: S3 ACLs and Bucket Policies are used to grant or deny permissions to S3 resources. Bucket Policies are attached at the bucket level and control access to the entire bucket, while ACLs are attached to individual objects and allow fine-grained control over access to those objects.

How can you monitor S3 usage and performance, and what AWS services can you use for this purpose?

Answer: You can monitor S3 usage and performance using AWS CloudWatch for metrics like data transfer, request rates, and bucket size. You can also use S3 access logs for detailed information on object access, including who accessed the data and when.

If one application is down in AWS what steps we have to check?

**1. Verify the Nature of the Outage**

* Check if the issue is affecting the entire application, specific instances, or only certain functionalities.
* Identify if the application is unresponsive, experiencing slow performance, or throwing specific errors.

**2. Check Application Logs**

* Access logs in AWS CloudWatch for error messages, crash reports, or unusual activity.
* Review recent changes in the code, configuration, or deployment.

**3. Examine CloudWatch Alarms and Metrics**

* Look for any CloudWatch alarms related to CPU, memory, disk space, or network usage.
* High CPU or memory usage can indicate a need to scale up resources.

**4. Review EC2 Health**

* Check the status of your EC2 instances to ensure they are running and accessible.
* Review the instance health status in the EC2 console and the System Status Checks (like network and hardware).
* Restart the instance if necessary and permissible.

**5. Check Auto Scaling and Load Balancer**

* If you are using Auto Scaling, verify if new instances have been provisioned in response to the issue.
* Check the health of the load balancer and whether it is correctly distributing traffic.

**6. Verify Network Configuration**

* Check VPC settings, security groups, and NACLs to confirm that there haven’t been changes impacting accessibility.
* Review route tables, subnets, and any possible connectivity issues between services.

**7. Database and Cache Health**

* For applications relying on databases (RDS, DynamoDB, etc.) or caching (Elasticache), check their health.
* Look at their CPU, memory, and network utilization. Check for connection issues or performance bottlenecks.

**8. Check for IAM or Permission Issues**

* Confirm that there haven’t been changes to IAM policies or roles that might affect access permissions.

**9. Review Recent Changes or Deployments**

* Confirm if there were any recent deployments, configuration changes, or infrastructure updates that might have introduced the issue.
* Rollback recent changes if they are suspected to be the cause.

**10. Check for Regional or Service-Specific Issues**

* Use the [AWS Service Health Dashboard](https://status.aws.amazon.com/) to check if there’s a known outage in your region.
* Look at AWS Personal Health Dashboard if you have access to it for any service issues specific to your account.

**11. Verify DNS and Domain Issues**

* Ensure that DNS routing is correctly configured if using Route 53 or other DNS providers.
* Check for domain expiration, SSL certificate issues, or DNS propagation delays if recent changes were made.

**12. Consider Restarting the Application or Components**

* Restart affected services or the application itself if you’ve pinpointed the source of the issue.

**13. Contact AWS Support**

* If you are unable to resolve the issue, reach out to AWS Support for assistance, providing them with details of your findings.

If EC2 is not running

**✅ 1. Check EC2 Instance State**

* **Console → EC2 → Instances**
  + Ensure the instance state is **"running"**.
  + If it's **"stopped"** or **"terminated"**, start or investigate accordingly.

**✅ 2. Check System Status Checks**

* Go to **EC2 → Instances → Status Checks**
  + There are two checks:
    - **System status check** – AWS-side issue (like hardware failure)
    - **Instance status check** – OS-side issue (boot errors, file system issues)
  + If **failing**, consider:
    - **Stop and start** the instance
    - Check **system logs**
    - Use **EC2 Serial Console** (for Nitro-based instances)

**✅ 3. Access the System Logs**

* **Actions → Monitor and troubleshoot → Get system log**
  + Look for:
    - Boot errors
    - Kernel panics
    - File system issues

**✅ 4. Check Network Configuration**

* **Security Groups:**
  + Ensure **inbound rules** allow:
    - **SSH (port 22)** for Linux
    - **RDP (port 3389)** for Windows
    - **HTTP/HTTPS** (80/443) if it's a web app
* **Network ACLs:**
  + Make sure they allow the necessary traffic
* **Elastic IP (if applicable):**
  + Make sure it is properly attached
* **Public IP:**
  + If the instance is in a public subnet, verify it has a public IP

**✅ 5. Check Subnet and Route Table**

* If you're using a **VPC**, ensure:
  + Subnet has a **route to an internet gateway**
  + The instance is in a **public subnet** (with public IP)
  + **NAT Gateway** is configured correctly (for private subnets)

**✅ 6. Try SSH or RDP Access**

* SSH into the instance using:

ssh -i your-key.pem ec2-user@<public-ip>

* If SSH fails:
  + Ensure your **.pem key** is correct
  + Check **security group rules**
  + Use **EC2 Instance Connect** if key is lost (for Amazon Linux)

**✅ 7. Use EC2 Instance Connect (for Amazon Linux 2 / Ubuntu)**

* From AWS Console → EC2 → Select Instance → **Connect → EC2 Instance Connect**
* If that fails, it confirms a **deeper OS-level issue**

**✅ 8. Review Application Logs (if instance boots but app doesn’t load)**

* Log in via SSH
* Check app logs:
  + Web servers: /var/log/httpd/, /var/log/nginx/
  + System logs: /var/log/messages, /var/log/syslog
* Confirm services are running (e.g., Apache, Nginx, etc.)

**✅ 9. Recovery Options**

If the instance is unreachable or corrupted:

* **Stop instance**, detach the root volume
* Attach it to another healthy instance as a secondary volume
* Mount and investigate the logs/config
* Fix any issues (e.g., corrupted fstab, bad configurations)
* Reattach it as root and restart the instance

**✅ 10. Check Billing and Quotas**

* Ensure your AWS account is not restricted due to billing issues
* Check **Service Quotas** (EC2 limits per region)

Ec2 instance is reboot multiple times automatically what troubleshoot we have to doTop of Form

**✅ 1. Check CloudTrail for Reboot Events**

Determine **who or what initiated the reboot**.

aws cloudtrail lookup-events --lookup-attributes AttributeKey=ResourceName,AttributeValue=<instance-id>

* Look for:
  + RebootInstances
  + StopInstances
  + StartInstances
* If the UserIdentity is an AWS service, automation, or a specific user, that can narrow things down.

**✅ 2. Review Instance System Logs**

Go to the EC2 Console:

* Select the instance → **Actions → Monitor and troubleshoot → Get system log**
* Look for:
  + Kernel panics
  + Filesystem errors
  + Boot loop messages
  + OOM (Out of Memory) errors

**✅ 3. Check CloudWatch Logs and Metrics**

* Navigate to **CloudWatch → Logs/Alarms**
* Look at:
  + CPU utilization spikes
  + Memory pressure (if custom metrics)
  + Disk IO
  + **Auto Recovery** events
  + **Reboot count** in logs

**✅ 4. Check Auto-Recovery and Auto-Scaling Events**

**🔹 Auto-Recovery (enabled via instance settings or alarms)**

aws ec2 describe-instance-status --instance-id <instance-id>

Check for:

"Events": [

{

"Code": "instance-reboot",

"Description": "Instance reboot due to recovery..."

}

]

**🔹 Auto Scaling Group?**

* If the instance is in an ASG (Auto Scaling Group), it may be **terminated and replaced** due to failing health checks.

**✅ 5. Check for Scheduled Events**

aws ec2 describe-instance-status --instance-id <instance-id> --include-all-instances

Look for:

* "Events" like system-reboot, instance-stop, or instance-retirement

**✅ 6. Inspect CloudWatch Alarms**

* Go to CloudWatch → Alarms
* Check if any alarm is triggering a reboot action via **SNS or Lambda**

**✅ 7. Check Crontab and Init Scripts**

SSH into the instance and run:

bash

CopyEdit

crontab -l

sudo crontab -l

cat /etc/crontab

Look for any reboot commands:

bash

CopyEdit

@reboot /sbin/reboot

Also check:

bash

CopyEdit

ls -l /etc/init.d/

ls -l /etc/systemd/system/

**✅ 8. Check for Malware or Corrupt Software**

* Look at unexpected processes running (top, ps aux)
* Check startup scripts for unauthorized changes
* Run antivirus scan (e.g., ClamAV for Linux)

**✅ 9. Check Disk Space and Memory**

bash

CopyEdit

df -h

free -m

dmesg | grep -i oom

* A full disk or memory exhaustion can cause reboot loops

**✅ 10. Check Application Logs**

* If an app or agent (e.g., custom monitoring or patching agent) triggers reboots, it should log them
* Look at /var/log/, especially:
  + /var/log/messages
  + /var/log/syslog
  + /var/log/cloud-init.log

**✅ 11. Test with a New Instance**

* Create a new EC2 instance with the same AMI or EBS volume
* See if the issue persists — if yes, it's likely OS or application-level

**Summary**

| **Check** | **Purpose** |
| --- | --- |
| CloudTrail | See who/what rebooted it |
| System Logs | Detect OS/kernel level issues |
| CloudWatch | Review metrics and alarms |
| Crontab & Init Scripts | Check scheduled reboot scripts |
| Auto Recovery / ASG | AWS might be auto-recovering or replacing instance |
| Malware | Rule out security issues |
| Disk/Memory Logs | Identify OOM or FS errors |

Bottom of Form

AWS Lambda

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS) that lets you run code without provisioning or managing servers. It automatically executes your code in response to specific triggers, such as changes in data, HTTP requests, or scheduled events